An Adaptive Policy for Improved Timeliness in Secure Database Systems

Sang H. Son*, Rasikan David*, and Bhavani Thuraisingham†

* Dept. of Computer Science, University of Virginia, Charlottesville, VA 22903, USA

† Mitre Corporation, Bedford, MA 01730, USA

Abstract

Database systems for real-time applications must satisfy timing constraints associated with transactions, in addition to maintaining data consistency. In addition to real-time requirements, security is usually required in many applications. Multilevel security requirements introduce a new dimension to transaction processing in real-time database systems. In this paper, we argue that because of the complexities involved, trade-offs need to be made between security and timeliness. We first describe a secure two-phase locking protocol. The protocol is then modified to support an adaptive method of trading off security for timeliness, depending on the current state of the system. The performance of the Adaptive 2PL protocol is evaluated for a spectrum of security-factor values ranging from fully secure (1.0) right upto fully real-time (0.0).

Keywords

Timeliness, concurrency control, two-phase locking, non-interference, security, miss percentage

1 INTRODUCTION

Database security is concerned with the ability of a database management system to enforce a security policy governing the disclosure, modification or destruction of information. Most secure database systems use an access control mechanism based on the Bell-LaPadula model [Bell76]. This model is stated in terms of subjects and objects. An object is understood to be a data file, record or a field within a record. A subject is an active process that requests access to objects.

This work was supported in part by ONR.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1995		**00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Adaptive Policy for Improved Timeliness in Secure Database Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Virginia,Department of Computer Science,151 Engineer's Way,Cahrlottesville,VA,22094-4740**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**15**	

Every object is assigned a classification and every subject a clearance. Classifications and clearances are collectively referred to as security classes (or levels) and they are partially ordered. The Bell-LaPadula model imposes the following restrictions on all data accesses:

a) *Simple Security Property*: A subject is allowed read access to an object only if the former's clearance is identical to or higher (in the partial order) than the latter's classification.

b) *The *-Property*: A subject is allowed write access to an object only if the former's clearance is identical to or lower than the latter's classification.

Database systems that support the Bell-LaPadula properties are called multilevel secure database systems (MLS/DBMS). The Bell-LaPadula model prevents direct flow of information from a higher access class to a lower access class, but the conditions are not sufficient to ensure that security is not violated indirectly through what are known as covert channels [Lamp73]. A covert channel allows indirect transfer of information from a subject at a higher access class to a subject at a lower access class. An important class of covert channels that are usually associated with concurrency control mechanisms are timing channels. A timing channel arises when a resource or object in the database is shared between subjects with different access classes. The two subjects can cooperate with each other to transfer information.

A real time database management system (RTDBMS) is a transaction processing system where transactions have explicit timing constraints. Typically a timing constraint is expressed in the form of a deadline, a certain time in the future by which a transaction needs to be completed. As advanced database systems are being used in applications which need to support timeliness while managing sensitive information, one cannot avoid the need for integrating real-time data processing techniques into MLS/DBMS. In this paper, we discuss the issues that arise when transactions in a secure database have timing constraints associated with them. For certain applications in which absolute security is required for safety-critical operations, any trade-offs of security for timeliness cannot be allowed. The approach presented in this paper is not intended to support such applications.

2 BACKGROUND

2.1 Correctness Criteria for Secure Scheduler

Covert channel analysis and removal is one of the most important issues in multilevel secure concurrency control. The notion of *non-interference* has been proposed [Gogu82] as a simple and intuitively satisfying definition of what it means for a system to be secure. The property of *non-interference* states that the output as seen by a subject must be unaffected by the inputs of another subject at a higher access class. This means that a subject at a lower access class should not be able to distinguish between the outputs from the system in response to an input sequence including actions from a higher level subject and an input sequence in which all inputs at a higher access class have been removed.

An extensive analysis of the possible covert channels in a secure concurrency control mecha-

nism and the necessary and sufficient conditions for a secure, interference-free scheduler are given in [Keef90]. Three of these properties are of relevance to the secure two phase locking protocol discussed in this paper. For the following definitions, given a schedule *s* and an access level *l, purge(s,l)* is the schedule with all actions at a level > *l* removed from *s*.

1) *Value Security*: A scheduler satisfies this property if values read by a subject are not affected by actions with higher subject classification levels. Stated formally, for an input schedule *p*, the output schedule *s* is said to be *value secure* if *purge(s,l)* is view equivalent to the output schedule produced for *purge(p,l)*. Two schedules are said to be view equivalent if each transaction's read operations read the same value, and for each data object, the final write is the same in both schedules.

2) *Delay Security*: This property ensures that the delay experienced by an action is not affected by the actions of a subject at a higher classification level. For an input schedule *p* and an output schedule *s*, a scheduler is *delay secure* if for all levels *l* in *p*, each of the actions a_1 in *purge(p,l)* is delayed in the output schedule produced for *purge(p,l)* if and only if it is delayed in *purge(s,l)*.

3) *Recovery Security*: A system ensures that the occurrence of a deadlock appears the same to a low-level subject, independent of whether higher level actions are in the schedule or not. The actions taken to recover from deadlock are also not affected by the presence of higher level transactions.

2.2 Approaches to Secure Concurrency Control

Locking will fail in a secure database because the security properties prevent actions in a transaction T_1 at a higher access class from delaying actions in a transaction T_2 at a lower access class (e.g. when T_2 requests a conflicting lock on a data item on which T_1 holds a lock). Timestamp ordering fails for similar reasons, with timestamps taking the role of locks, since a transaction at a higher access class cannot cause the aborting of another transaction at a lower access class. Locking and timestamping techniques can be adapted for secure DBMS.

Optimistic concurrency control for a secure database can be made to work by ensuring that whenever a conflict is detected between a transaction T_h at a higher access class in its validation phase and a transaction T_l at a lower access class, the transaction at the higher access class is aborted, while the transaction at the lower access class is not affected. A major problem with using optimistic concurrency control is the possible *starvation* of higher-level transactions. For example, consider a long-running transaction T_h that must read several lower-level data items before the validation stage. In this case, there is a high probability of conflict and as a result, T_h may have to be rolled back and restarted an indefinite number of times.

Basic multiversion timestamp ordering (MVTO) can be extended to Secure MVTO. The difference between Basic MVTO and Secure MVTO is that Secure MVTO will assign a new transaction a timestamp that is earlier than the current timestamp. This effectively moves the transaction into the past with respect to active transactions. This approach to timestamp assignment is what makes it impossible for a transaction to invalidate a read from a higher access class.This method has the drawback that transactions at a higher access class are forced to read arbitrarily old values

from the database because of the timestamp assignment. This problem can be especially serious if most of the lower level transactions are long running transactions. Alternative approach is to make higher access class transaction wait until all transactions that are lower and have arrived earlier finish their execution.

3 SUPPORTING SECURITY AND TIMELINESS

There are several papers that have explored approaches to extend conventional databases for time-critical applications [Abbo92, Sha91, Son92]. The problem arises when these approaches are applied to secure databases, because covert channels can be introduced by priority-based scheduling. All existing real-time systems schedule transactions based on some priority scheme. The priority usually reflects how close the transaction is to missing its deadline. Priority-based scheduling of real-time transactions, however, interacts with the property of non-interference that has to be satisfied for security. For example, consider the following sequence of requests:

T_1 (SECRET)	:	*R(X)*					
T_2 (UNCLASSIFIED)	:		*W(X)*				
T_3 (UNCLASSIFIED)	:				*W(X)*		
T_4 (UNCLASSIFIED)	:						*R(X)*

Assume that T_1, T_2 and T_3 have priorities *5, 7* and *10* respectively and the priority assignment scheme is such that if *priority*(T_2) > *priority*(T_1), then T_2 has greater criticalness and has to be scheduled ahead of T_1. In the above example, T_2 and T_3 are initially blocked by T_1 when they arrive. When T_1 completes execution, T_3 is scheduled ahead of T_2, since it has a greater priority than T_2 and the transaction execution order would be T_1 T_3 T_2 T_4. However, if the transaction T_1 is removed, the execution order would be T_2 T_3 T_4 because T_2 would have been scheduled as soon as it had arrived. The presence of the SECRET transaction T_1 thus changes the value read by the UNCLASSIFIED transaction T_4, which is a violation of *value security.* For the same reason *delay security* is also violated, because the presence of T_1 delays T_2 with respect to T_3.

We use the secure two phase locking protocol [Son94] as a basis for improving the timeliness of secure database systems, because it showed the best average performance when compared to other protocols. Priority-based transaction scheduling is not feasible for a fully secure database system, because in a secure environment a transaction at a higher level:

- cannot cause the aborting of a transaction at a lower access class. If it is allowed to do so, it is possible that it can control the number of times a lower level transaction is aborted, thereby opening a covert channel.
- cannot conflict with a transaction at a lower access class. If such a conflict does occur, the higher level transaction has to be blocked or aborted, not the low level transaction.
- cannot be granted greater priority of execution over a transaction at a lower access class.

Therefore, for minimizing deadline miss percentage, we take the approach that partial security violations under certain conditions are permissible. In the subsequent sections, the Secure 2PL protocol is described and the design of a trade-off factor between the security properties and the real-time requirements is presented.

4 SECURE TWO-PHASE LOCKING

Basic two-phase locking does not work for secure databases because a transaction at a lower access class (say T_l) cannot be blocked because of a conflicting lock held by a transaction at a higher access class (T_h). If T_l were somehow allowed to continue with its execution in spite of the conflict, then non-interference would be satisfied. The basic principle behind the secure two-phase locking protocol is to try to simulate execution of Basic 2PL without blocking of lower access class transactions by higher access class transactions. Three different types of locks are used for this purpose. Their semantics are explained below:

1) *Real Lock (of the form $pl_i[x]$)*: A real lock is set for an action $p_i[x]$ if no other conflicting action has a real lock or a virtual lock on x. The semantics of this lock are identical to that of the lock in basic two phase locking.

2) *Virtual Lock (of the form $vpl_i[x]$)*: A virtual lock $vpl_i[x]$ is set for an action $p_i[x]$ if a transaction at a higher access class holds a conflicting lock on x ($p_i[x]$ has to be a write to satisfy the Bell-LaPadula properties). The virtual lock is non-blocking. Once a virtual lock $vpl_i[x]$ is set, $p_i[x]$ is added to *queue[x]* and the next action in T_i is ready for scheduling. When $p_i[x]$ gets to the front of the lock queue, its virtual lock is upgraded to a real lock and $p_i[x]$ is submitted to the scheduler. A virtual lock holding action $vpl_i[x]$ can be superseded in the lock queue by a conflicting action $q_j[x]$ if T_j is in *before(T_i)*.

3) *Dependent Virtual Lock (of the form $dvpl_i[x]$)*: A dependent virtual lock is set for an action $p_i[x]$ (where p is a write) if a previous write $w_i[y]$ in the same transaction holds a virtual lock. An action $p_i[x]$ which holds a dependent virtual lock with respect to another action $w_i[y]$ is not allowed to set a real lock or a virtual lock unless $w_i[y]$'s virtual lock is upgraded to a real lock. The dependent lock is non-blocking and can be superseded by a conflicting action $q_j[x]$ if T_j is before T_i in the serialization order.

4.1 An Example

Consider the two transactions in example 1:

T_1 (SECRET)	:	$r[x]$	•••	c_1
T_2 (UNCLASSIFIED)	:	$w[x]$	c_2	

EXAMPLE 1

Basic two phase locking would fail because $w_2[x]$ would be blocked waiting for T_1 to commit and release $rl_1[x]$. In our modification to the two-phase locking protocol, T_2 is allowed to set a virtual lock $vwl_2[x]$, write onto a version of x local to T_2 and continue with the execution of its next operation, i.e. c_2. When T_1 commits and releases the lock on x, T_2's virtual write lock is upgraded to a real lock and $w_2[x]$ is performed. Until $w_2[x]$ is performed, no conflicting action is allowed to set a lock on x. The sequence of operations performed is therefore, $rl_1[x]\ r_1[x]\ vwl_2[x]\ c_2$ ••• $c_1\ ru_1[x]\ wl_2[x]\ w_2[x]\ wu_2[x]$.

This modification alone is not enough, as illustrated in Example 2:

T_1 (SECRET)	:	$r[x]$		$r[y]$	c_1
T_2 (UNCLASSIFIED)	:		$w[x]$ $w[y]$		c_2

EXAMPLE 2

The sequence of operations that would be performed is $rl_1[x]$ $r_1[x]$ $vwl_2[x]$ $vw_2[x]$ $wl_2[y]$ $w_2[y]$ c_2. After these operations, deadlock would occur because $r_1[y]$ waits for $w_2[y]$ to release its lock and $vw_2[x]$ waits for $r_1[x]$ to release its lock. This deadlock would not have occurred in basic two-phase locking. Note that our aim of trying to simulate execution of basic two phase locking is not being achieved. On closer inspection, it is obvious that this problem arises because $w_2[y]$ is allowed to proceed with its execution even though $w_2[x]$ could only write onto a local version of x because of the read lock $rl_1[x]$ set by T_1. To avoid this problem, for each transaction T_i, two lists are maintained - *before*(T_i) which is the list of active transactions that precede T_i in the serialization order and *after*(T_i) which is the list of active transactions that follow T_i in the serialization order. The following additions are made to the basic two-phase locking protocol:

1) When an action $p_i[x]$ sets a virtual lock on x because of a real lock $ql_j[x]$ held by T_j, then T_i and all transactions in *after*(T_i) are added to *after*(T_j), T_j and all transactions in *before*(T_j) are added to *before*(T_i).

2) When an action $w_i[x]$ arrives and finds that a previous action $w_i[y]$ (for some data item y) has already set a virtual write lock $vwl_i[y]$, then a dependent lock $dvwl_i[x]$ is set with respect to $vwl_i[y]$.

3) When an action $p_i[x]$ arrives and finds that a conflicting virtual or dependent lock $vql_j[x]$ or $dvql_j[x]$ has been set by a transaction T_j which is in *after*(T_i), then $p_i[x]$ is allowed to set a lock on x and perform $p_i[x]$ in spite of the conflicting lock.

4) A dependent virtual lock $dvp_i[x]$, dependent on some action $q_i[y]$ is upgraded to a virtual lock when $vql_i[x]$ is upgraded to a real lock.

The maintenance of a serialization order and the presence of dependent locks are necessary to prevent uncontrolled acquisition of virtual locks by transactions at lower access classes. For Example 2, the sequence of operations that would now be performed is $rl_1[x]$ $r_1[x]$ $vwl_2[x]$ $dvwl_2[y]$ c_2 $rl_1[y]$ $r_1[y]$ c_1 $ru_1[x]$ $ru_1[y]$ $wl_2[x]$ $w_2[x]$ $wu_2[x]$ $wl_2[y]$ $w_2[y]$ $wu_2[y]$.

The conditions given above are necessary to ensure that Delay Security is satisfied, but Value Security could still be violated as shown in Example 3.

T_1 (TOP SECRET)	:	$r_1[y]$	$r_1[x]$	•••		c_1
T_2 (SECRET)	:				$r_2[z]$	c_2
T_3 (CLASSIFIED)	:	$w_3[x]$		$w_3[z]$		c_3
T_4 (UNCLASSIFIED)	:		$w_4[y]$	$w_4[z]$		c_4

EXAMPLE 3

When T_4 requests a write lock on *'y'*, T_1 already holds a real lock on *'y'*, i.e., T_1 is set before T_4 in the serialization order. When T_1 requests a read lock on *'x'*, T_3 already holds a write lock on *'x'*, i.e., T_3 is set before T_1 in the serialization order. When T_3 requests a write lock on *'z'*, T_4 already holds a dependent lock on *'z'*. However, T_3 is before T_1 in the serialization order and T_1 is, in turn, before T_4 in the serialization order, i.e., T_3 is before T_4 in the serialization order and so, $w_3[z]$ is allowed to supersede $w_4[z]$ in the lock queue for *'z'*. Now, when $r_2[z]$ is submitted, it will read the value of *'z'* written by T_4. However, if T_1 had not been present, then T_3 would not have been before T_4 in the serialization order. This means that $w_3[z]$ would not have superseded $w_4[z]$ in the lock queue for *'z'*, i.e., $r_2[z]$ would have read the value of *'z'* written by T_3. Since the value read by a SECRET transaction is affected by the presence of a transaction at the TOP SECRET level, Value Security is being violated.

In general, an action in transaction T_1 can supersede an action in transaction T_2 if T_1 is before T_2 in the serialization order. Now, Value Security could be violated if T_1 is before T_2 in the serialization order because of the presence of a transaction T_3 at a higher access class, i.e., if T_3 is between T_1 and T_2 in the serialization order. To overcome this problem, the action in T_1 is not allowed to supersede the action in T_2 if such a case arises.

4.2 The Secure Two-Phase Locking Protocol

In the subsequent description of the protocol, queue[x] is a queue of operations, each of which could be holding a lock (real, virtual, or dependent virtual) or waiting to set a lock (Wait). SL(Ti) is the security access class of transaction Ti.

When an action $p_i[x]$ is submitted, one of three possible cases can arise:

Case 1 $(p = read) \wedge \exists (wLock(T_i, x) \vee VwLock(T_i, x) \vee DVwLock(T_i, x))$

Read value of x written by $w_i[x]$;

Case 2 $((p = write) \wedge \exists rLock(T_i, x))$

Upgrade $rLock(T_i,x)$ to $wLock(T_i,x)$;

Case 3
$$\exists j\,((qLock(T_j, x) \vee VqLock(T_j, x) \vee DVqLock(T_j, x) \vee Wait(T_j, x)) \wedge (j \neq i) \wedge ((p = write) \vee (q = write)))$$

pos := length(queue[x]);
while (pos > 0)
$T_j \leftarrow$ transaction at $queue[x] \rightarrow pos$;
if (operation($queue[x] \rightarrow pos$) conflicts with p)
if $(T_i \in before(T_j))$
$$\forall T_3 ((T_3 \in before(Tj)) \wedge (T_3 \in after(T_i)) \wedge (SL(T_3) > SL(T_i)) \wedge (SL(T_3) > SL(T_j)))$$

```
                    abort T3; /* value security violation */
                if (lock(queue[x] → pos) is a real lock)    /* deadlock */
                    call victim selection routine;
                    if (victim = Ti)
                        exit;
                    else
                        continue;
            if (Ti ∉ before(Tj))
                if ((p = write) ∧ (∃z, VwLock(Ti, z)))
                    Insert(queue,DVwLock(Ti,x),pos); /* insert after pos */
                    dependent(wi[z]) := wi[x];
                else if (SL(Ti) < SL(Tj))
                    Insert(queue,VwLock(Ti,x),pos);
                else
                    Insert(queue,pWait(Ti,x),pos);
                before(Ti) := before(Ti) ∪ {Tj} ∪ before(Tj) ;
                after(Tj) := after(Tj) ∪ {Ti} ∪ after(Ti)
        pos := pos - 1;
    if (pos = 0)
        Insert(queue[x],pLock(Ti,x),pos);
```

In addition, the following two conditions are to be satisfied by the transaction manager:

1) A transaction is allowed to commit, even if virtual writes performed by the transaction have not been committed to stable storage, i.e the writes are still on some queue[x] waiting to set a real lock on x.

2) Once an action $p_i[x]$ acquires a lock on a data item, the lock is not released until T_i commits and $p_i[x]$ is performed on the data item x in stable storage. Therefore, in most cases, all the locks that a transaction holds are not released when it commits.

The correctness of the protocol is proved in [Son94]. The protocol satisfies the value security requirement by checking at the stage where it could possibly be violated. It also satisfies the delay security because an action at a lower access class is never delayed by an action at a higher class. Two additional problems that need to be addressed to make the protocol useful is the recovery from system failures and deadlock resolution. The recovery procedure is discussed in [Dav93].

4.3 Deadlock Detection and Resolution

The secure two-phase locking protocol is not free from deadlocks. For example, consider the input schedule:

T_1 (SECRET)	:	*r[x]*		*r[y]*	
T_2 (UNCLASSIFIED)	:		*w[y]*		*w[x]*

The sequence of operations performed are $rl_1[x]$ $r_1[x]$ $wl_2[y]$ $w_2[y]$ $vwl_2[x]$ c_2. After these operations, deadlock would occur because $r_1[y]$ waits for $w_2[y]$ to release its lock and $vw_2[x]$ waits for $r_1[x]$ to release its lock. Deadlocks can be detected by constructing a wait-for graph [Bern87]. A scheduler recovers from a deadlock by aborting one of the transactions in the cycle. A secure scheduler must ensure that Recovery Security is not violated by the choice of a victim. This can be guaranteed by aborting a transaction at the highest access class in the cycle detected in the wait-for graph.

5 AN ADAPTIVE SECURITY POLICY

The Secure 2PL exhibits a much better response time characteristic than Secure OCC. Its operating region (the portion of the curve before the saturation point) is much larger than that of Secure OCC (Figure 1). Further, staleness is not an issue in Secure 2PL as with Secure MVTO. However, this alone does not suffice when timing constraints are present on transactions. In Secure 2PL, transaction scheduling order is determined purely by the order in which transactions acquire locks. No conscious effort is made to schedule transactions according to their priority, or according to how close a transaction is to meeting its deadline. In a real-time database system this is unacceptable. In Section 3, we have seen that priority-driven scheduling of transactions could lead to security violations. It is our claim that the security properties have to be sacrificed to some extent to ensure a certain degree of deadline cognizance.

The *bandwidth* of a covert channel is a measure of how easy it is for the higher access class transaction to control the delay seen by the lower access class transaction. If there is a great degree of randomness in the system, i.e., an indeterminate number of transactions could be affecting the delay that the higher access class transactions wants a lower access class transaction to experience, then the bandwidth is low. On the other hand, if the higher access class transaction knows that the lower access class transaction to which it wants to transmit information is the only other transaction in the system, then the bandwidth is infinite. Therefore, when security has to be sacrificed, a policy that keeps the bandwidth of the resulting covert channel to a minimum is desirable. To ensure this, the security policy has to be adaptive, i.e., determining whether security is to be violated or not when a conflict arises should depend on the current state of the system and not on a static, predecided property.

Our adaptive policy to resolve conflicts between lock holding and lock requesting transactions is based on past execution history. Whenever a transaction T_1 requests a lock on a data item x on which another transaction T_2 holds a conflicting lock, there are two possible options:

- T_1 could be blocked until T_2 releases the lock.
- T_2 could be aborted and the lock granted to T_1.

If T_1 were at a higher security level than T_2, the latter option would be a violation of security.

However, if T_1 has greater priority than T_2, then the latter option would be the option taken by a real-time concurrency control approach. In our approach, we strike a balance between these two conflicting options by looking up past history. A measure of the degree to which security has been violated in the past is calculated. A similar measure of the degree to which the real-time constraints have not been satisfied can be obtained from the number of deadlines missed in the past. These two measures are compared and depending on which value is greater, either the security properties are satisfied or the higher priority transaction is given the right to execute.

The two factors that are used to resolve a conflict are:

- Security Factor (SF): (number of conflicts for which security is maintained / total number of conflicts) * difference in security level between conflicting transactions.

- Deadline Miss Factor (DMF): number of transactions that missed their deadline / total number of transactions committed

Two factors are involved in the calculation of SF. The first factor is the degree to which security has been satisfied in the past, measured by the number of conflicts for which security has been maintained. Secondly, we also assume that the greater the difference in security levels between the transactions involved in the conflict, the more important it is to maintain security. DMF is determined only by the number of deadline misses in the past. Note that for a comparison with DMF, (1 - SF) has to be used, since (1 - SF) is a measure of the degree to which security has been violated. Now, a simple comparison (1 - SF) > DMF is not enough, since different systems need to maintain different levels of security. Therefore, we define two weighting factors, α and β for (1 - SF) and DMF respectively. If $\alpha * (1 - SF) > \beta * DMF$, then for the conflict under consideration, the security properties are more important and therefore the conflict is decided in favor of the transaction at a lower access class. If the opposite is true, then the transaction with higher priority is given precedence. Note that at low conflict rates, it is possible to satisfy both the security and the real-time requirements simultaneously. As a result the comparison is not made until the DMF reaches a certain threshold value DMISS_THRESH. The parameters DMISS_THRESH, α and β can be tuned for the desired level of security. A very high value of DMISS_THRESH or a very high value of α compared to β would result in SF being maintained at 1.0, i.e., for all conflicts the security properties are satisfied. A very high value of β compared to α would result in an SF value of 0.0, i.e., the behavior would be identical to that of 2PL-HP [Abbo92]. For a desired value of SF between 0 and 1, the values of α, β and DMISS_THRESH would have to be tuned based on the arrival rate of transactions.

The hybrid protocol is defined by the following rules:

If a conflict between a lock holding transaction T_1 and a lock requesting transaction T_2 arises, the conflict is settled using the following rules:

- *If DMF < DMISS_THRESH then follow the steps taken by the Secure 2PL protocol*
- *Else If α * (1 - SF) > β * DMF, follow the steps taken by the Secure 2PL protocol*
- *Else break the conflict in favor of the transaction with the higher priority*

6 PERFORMANCE EVALUATION

In this section, we present the results of our performance study of the Adaptive Secure 2PL protocol for a wide range of values of SF. The two main goals of our performance analysis are:

- To determine miss percentages for varying transaction arrival rates for various values of SF.
- Since our model assumes a soft deadline system, the second factor that has been measured is tardy time - the difference between the commitment time and deadline for late transactions.

6.1 Simulation Model

Central to the simulation model is a single-site disk resident database system operating on shared-memory multiprocessors. The system consists of a disk-based database and a main memory cache. The unit of database granularity is the *page*. When a transaction needs to perform an operation on a data item it accesses a page. If the page is not found in the cache, it is read from disk. CPU or disk access is through an M/M/k queueing system, consisting of a single queue with *'k'* servers (where *'k'* is the number of disks or CPUs).

In the model for Adaptive Secure 2PL, the execution of a transaction consists of multiple instances of alternating data access requests and data operation steps, until all the data operations in it complete or it is aborted. When a transaction makes a data request, i.e., lock request on a data object, the request must go through concurrency control to obtain a lock on the data object. If $\alpha * (1 - SF) < \beta * DMF$, then if the transaction's priority is greater than all of the lock holders, the holders are aborted and the transaction is granted a lock; if the transaction's priority is lower, it waits for the lock holders to release the lock [Abbo92]. However, if $\alpha * (1 - SF) > \beta * DMF$, then the steps taken by the Secure 2PL are followed. If the request for a lock is granted, the transaction proceeds to perform the data operation, which consists of a possible disk access (if the data item is not present in the cache) followed by CPU computation. However, if only a virtual or dependent lock is granted, the transaction only does CPU computation, since the operation should only be performed on a local version. If the request for the lock is denied (the transaction is blocked), the transaction is placed into the data queue. When the waiting transaction is granted a lock, only then can it perform its data operation. Also, when a virtual lock for an operation is upgraded to a real lock, the data operation requires disk access and CPU computation. At any stage, if a deadlock is detected, the transaction to be aborted to break the deadlock is determined, aborted and restarted. When all the operations in a transaction are completed, the transaction commits. Even if a transaction misses its deadline, it is allowed to execute until all its actions are completed.

6.2 Parameters and Performance Metrics

Table 1 gives the names and meanings of the parameters that control system resources. The parameters, *CPUTime* and *DiskTime* capture the CPU and disk processing times per data page. Our simulation system does not explicitly account for the time needed for data operation scheduling. We assume that these costs are included in *CPUTime* on a per-data-object basis. The use of a database cache is simulated using probability. When a transaction attempts to read a data page, the system determines whether the page is in cache or disk using the probability *BufProb*. If the

page is determined to be in cache, the transaction can continue processing without disk access. Otherwise disk access is needed.

Table 1: System Resource Parameters

Parameter	Meaning	Base Value
DBSize	Number of data pages in database	350
NumCPUs	Number of processors	2
NumDisks	Number of disks	4
CPUTime	CPU time for processing an action	15 msec
DiskTime	Disk service time for an action	25 msec
BufProb	Prob. of a page in memory buffer	0.5
NumSecLevels	Num. of security levels supported	6

Table 2 summarizes the key parameters that characterize system workload and transactions. Transactions arrive in a Poisson stream, i.e., their inter-arrival rates are exponentially distributed. The *ArriRate* parameter specifies the mean rate of transaction arrivals. The number of data objects accessed by a transaction is determined by a normal distribution with mean *TranSize*, and the actual data objects to be accessed are determined uniformly from the database.

Table 2: Workload Parameters

Parameter	Meaning	Base Value
ArriRate	Mean transaction arrival rate	-
TranSize	Average Transaction Size	6
RestartDelay	Mean overhead in restarting	1 msec
MinSlack	Minimum slack factor	2
MaxSlack	Maximum slack factor	8

The assignment of deadlines to transactions is controlled by the parameters *MinSlack* and *MaxSlack*, which set a lower and upper bound, respectively, on a transaction's slack time. We use the formula for deadline-assignment to a transaction.

Deadline = AT + Uniform(*MinSlack*,*MaxSlack*) * *ET*

AT and ET denoting the arrival time and execution time, respectively. The execution time of a transaction used in this formula is not an actual execution time, but a time estimated using the values of parameters *TranSize*, *CPUTime* and *DiskTime*. The priorities of transactions are decided by the *Earliest Deadline First* policy.

The primary performance metric used is miss percentage, which is the ratio of the number of transactions that do not meet their deadline to the total number of transactions committed. The second performance metric is tardy time, which indicates the average amount of time to finish the transaction beyond their deadlines.

6.3 Experiments and Results

For each experiment, we ran the simulation with the same parameters for 6 different random number seeds. Each simulation run was continued until 200 transactions at each access class were committed. For each run, the statistics gathered during the first few seconds were discarded in order to let the system stabilize after an initial transient condition. For each experiment the performance measure was measured over a wide range of workload. All the data reported in this paper have 90% confidence intervals, whose endpoints are within 10% of the point estimate.

6.3.1 Experiment 1: Secure 2PL vs Others

The first experiment was conducted to compare the average case performance of Secure 2PL. Note that good average case performance is also essential for a real-time system. A slow system cannot, however, be expected to meet timing constraints. In this experiment, the response times of Secure 2PL, Secure OCC and Secure MVTO at each security level were measured for varying arrival rates. The resulting graphs are shown in Figures 1 and 2. At low arrival rates, the response times are more or less the same for all three approaches. This is because the contention levels are low and majority of time is spent in disk access and CPU access rather than in resource queues, lock queues or transaction aborts. As the arrival rate increases the impact of these factors increases, and depending on how much they increase in each concurrency control approach, the performance varies. In Secure OCC, the saturation point is reached much earlier than in Secure 2PL and Secure MVTO. As the arrival rate increases, the contention level for data items increases. As a result, when a higher access class transaction reaches its validation stage, invariably it would have conflicted with a transaction at a lower access class and would therefore have to be aborted and restarted. It was found that the system reached a stage where, at the highest access class, transactions were repeatedly being aborted and restarted, resulting in the steep increase in response time at the saturation point. Now, since transactions are not being committed while arrival rate is unchanged, the net number of active transactions keeps increasing, increasing contention for the finite resources. It is because of this phenomenon that the response time for lower access class transactions increases also at the saturation point. The performance margin of Secure 2PL at level 0 over Secure 2PL at level 5 is small, indicating a greater measure of fairness than in Secure OCC. In addition, the saturation point is reached at a much higher arrival rate in Secure 2PL than in Secure OCC. Of course, the response time graph for Secure MVTO is the best, since no transactions are blocked or aborted. The performance margin of Secure MVTO at level 0 over Secure MVTO at level 5 is negligibly small. The rate of increase in response time after the saturation point is also much less than that of both Secure OCC and Secure 2PL. How-

ever, in Secure MVTO, the good response time is offset by an unacceptable *staleness* factor. This is all the more critical in a real-time environment, where data could have temporal constraints associated with them.

6.3.2 Experiment 2: Adaptive Secure 2PL - Miss Percentage

In this experiment, the miss percentages for Adaptive Secure 2PL were measured for three different settings of the SF values - fully secure (SF = 1.0), no security (SF = 0.0) and partially secure (SF = 0.5), as shown in Figure 3. Since we are considering a real-time database system, we restrict attention to the portion of the graph where miss percentages are less than 10%. The performance after the saturation point is not an issue. As expected, 2PL-HP has the lowest miss percentage, with the curve for 2PL (SF=0.5) falling between 2PL-HP and Secure 2PL.

6.3.3 Experiment 3: Adaptive Secure 2PL - Tardy Time

In this experiment, the average tardy times for Adaptive Secure 2PL were measured for three different settings of the SF values - fully secure (SF = 1.0), no security (SF = 0.0) and partially secure (SF = 0.5). The resulting graph is shown in Figure 4. The results obtained confirm the results obtained from the miss percentage experiment. The tardy time is maximum for Secure 2PL, with the curve for Adaptive 2PL (SF = 0.5) falling between that of Secure 2PL and 2PL-HP.

7. CONCLUSIONS

In this paper, we have presented an approach to scheduling transactions to meet their timing constraints in a secure database. Our simulation results substantiate our claim that an adaptive security policy that sacrifices the security properties to some extent can significantly improve the deadline miss performance. Although the improvement is only a few percentage, making few more deadlines may make significant difference in overload situations in real-time applications. The graphs illustrate that the performance improvement is getting bigger as the system load increases.

The work described in this paper is more a direction for future research than a concrete solution to the problem of secure real-time concurrency control. There are a number of issues that need to be looked into. First of all, a proper characterization of the bandwidth of a covert channel that can arise given a particular value of SF needs to be derived. Applications might express a desired level of security in terms of a maximum admissible bandwidth of a potential covert channel. Unless there is a way of determining to what extent a security policy satisfies the security properties, one cannot determine whether the policy is suitable for the application or not. Secondly, in this paper we have considered a simple trade-off between deadline miss percentage and security. A trade-off could also have been made between alternative factors depending on the application. Thirdly, we have restricted ourselves to a soft deadline system with no overload management policy. It would be interesting to see how a policy to screen out transactions that are about to miss their deadline would affect performance. Finally, in this paper, we have restricted ourselves to the problem of real-time secure concurrency control in a database system. Some of the other issues

that need to be considered in designing a comprehensive real-time multilevel secure database system (MLS/RTDBMS) are dealt with in [Son93]. Various types of MLS/RTDBMSs need to be identified and architectures and algorithms developed for each type of system. Trade-offs need to be made between security, timeliness and consistency on a case-by-case basis.

REFERENCES

[Abbo92] R. K. Abbott and H. Garcia-Molina. "Scheduling Real-Time Transactions: A Performance Evaluation", ACM Trans. on Database Systems, 17(3), pp 513-560, Sept.1992.

[Bell76] D. E. Bell and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretation", The Mitre Corp., March 1976.

[Bern87] P. A. Bernstein, N. Goodman & V. Hadzilacos. "Concurrency Control and Recovery in Database Systems", Reading, MA: Addison Wesley, 1987.

[Dav93] R. David & S. H. Son. "A Secure Two Phase Locking Protocol", Proc. of the 12th Symposium on Reliable Distributed Systems, Princeton, NJ, Oct. 1993.

[Gogu82] J. A. Goguen and J. Meseguer. "Security Policy and Security Models", Proc. of the IEEE Symposium on Security and Privacy, pp 11-20, 1982.

[Keef90] T. Keefe, W. T. Tsai & J. Srivastava. "Multilevel Secure Database Concurrency Control", Proc. of Data Engineering Conference, pp 337-344, Los Angeles, CA, Feb. 1990.

[Lamp73] Butler W. Lampson. "A Note on the Confinement Problem", Communications of the ACM, 16(10), pp 613-615, October 1973.

[Sha91] Sha L., R. Rajkumar, S. H. Son, and C. Chang. "A Real-Time Locking Protocol", IEEE Trans. on Computers, 40(7), pp 782-800, July 1991.

[Son92] S. H. Son, J. Lee, and Y. Lin. "Hybrid Protocols Using Dynamic Adjustment of Serialization Order for Real-Time Concurrency Control", Real-Time Systems Journal, 4(3), pp 269-276, Sept. 1992.

[Son93] S. H. Son and B. Thuraisingham. "Towards a Multilevel Secure Database Management System for Real-Time Applications", IEEE Workshop on Real-Time Applications, New York, New York, May 1993.

[Son94] S. H. Son and R. David. "Design and Analysis of a Secure Two Phase Locking Protocol", International Computer Software and Applications Conference (COMPSAC'94), Taipei, Taiwan, pp 374-379, November 1994.

BIOGRAPHY

Sang H. Son received the Ph.D. in computer science from University of Maryland, and is currently an Associate Professor in the Department of Computer Science at the University of Virginia. His current research interests include real-time computing, database systems, and system security. He has served on numerous program committees of international conferences in those areas. He edited the book "Advances in Real-Time Systems," published by Prentice-Hall in 1995.

Rasikan David received the MS degree in computer science from University of Virginia in 1994.

Bhavani Thuraisingham is a principal engineer with MITRE Corporation. She received the Ph.D. from the University of Wales, U.K. Her research interests include database security and multimedia databases. Before joining MITRE, she worked at Honeywell and Control Data Corporation.